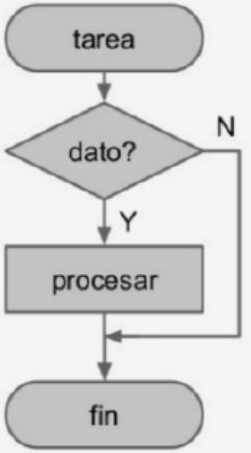
Sistemas RTOS

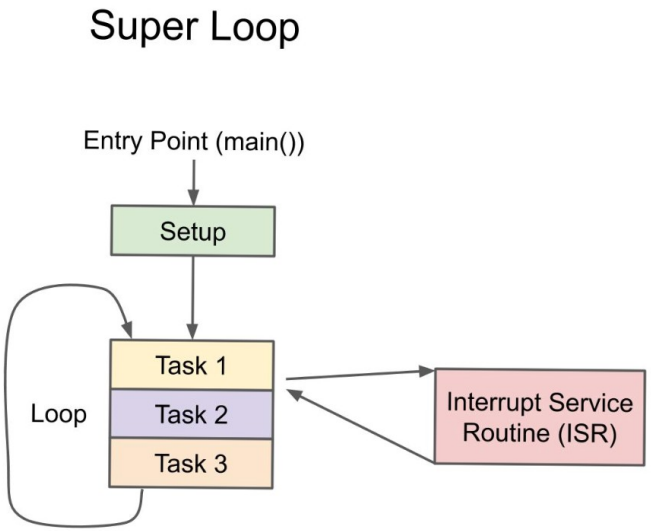
Programación avanzada– Ingeniería Mecatrónica - **2025**

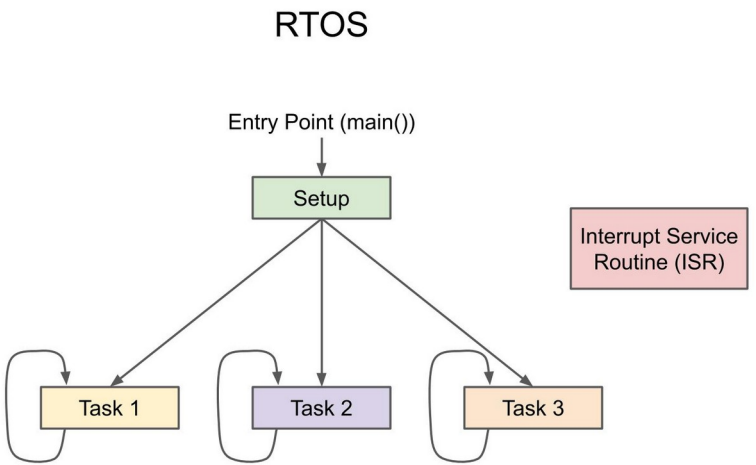
Modelo de programacion bare-metal (super loop)

```
while(1)
{
   tarea1();
   tarea2();
   tarea3();
   ...
}
```

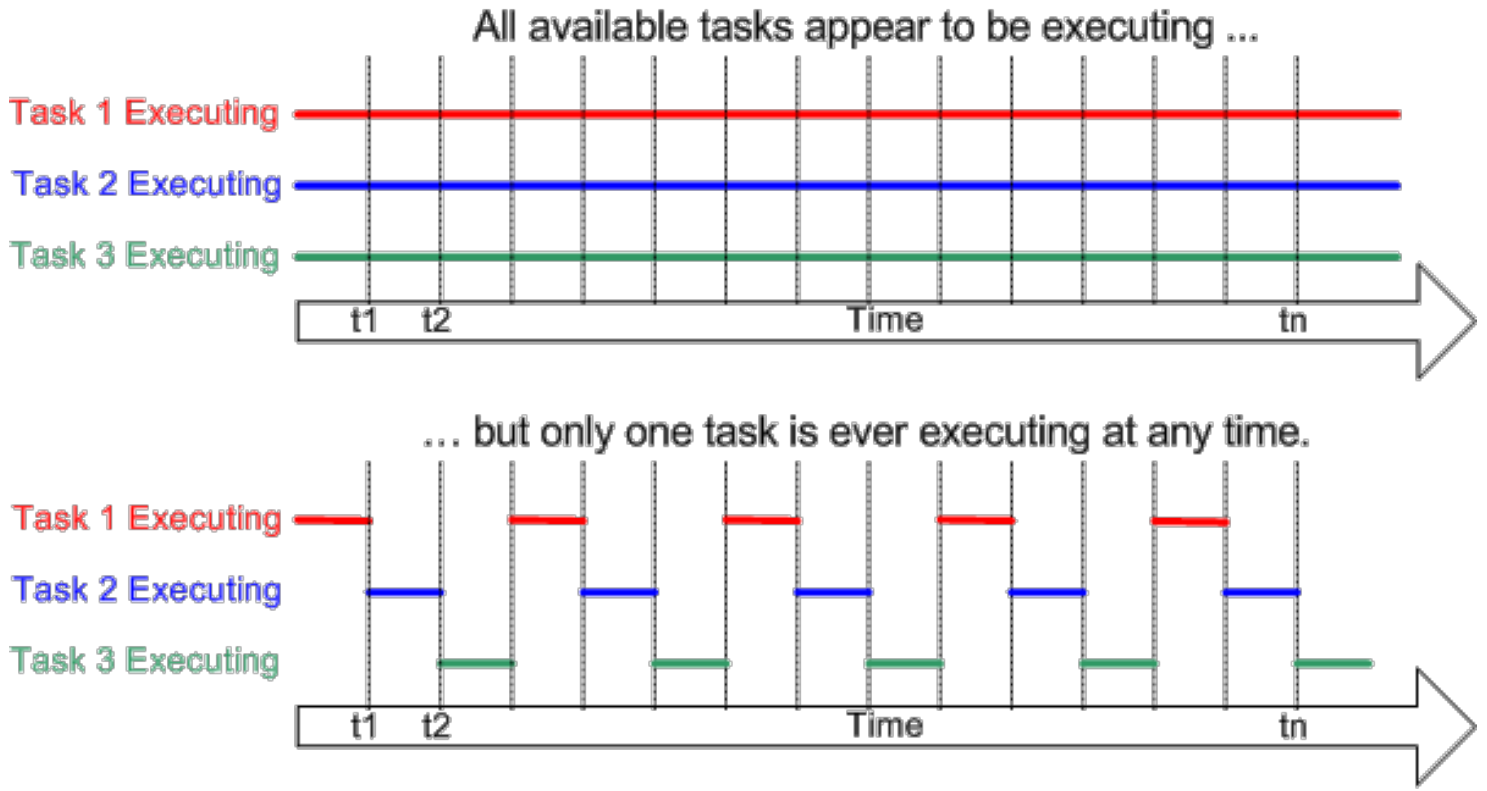


Comparacion

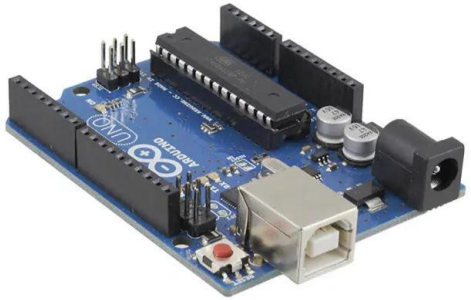




Comparacion



hardware



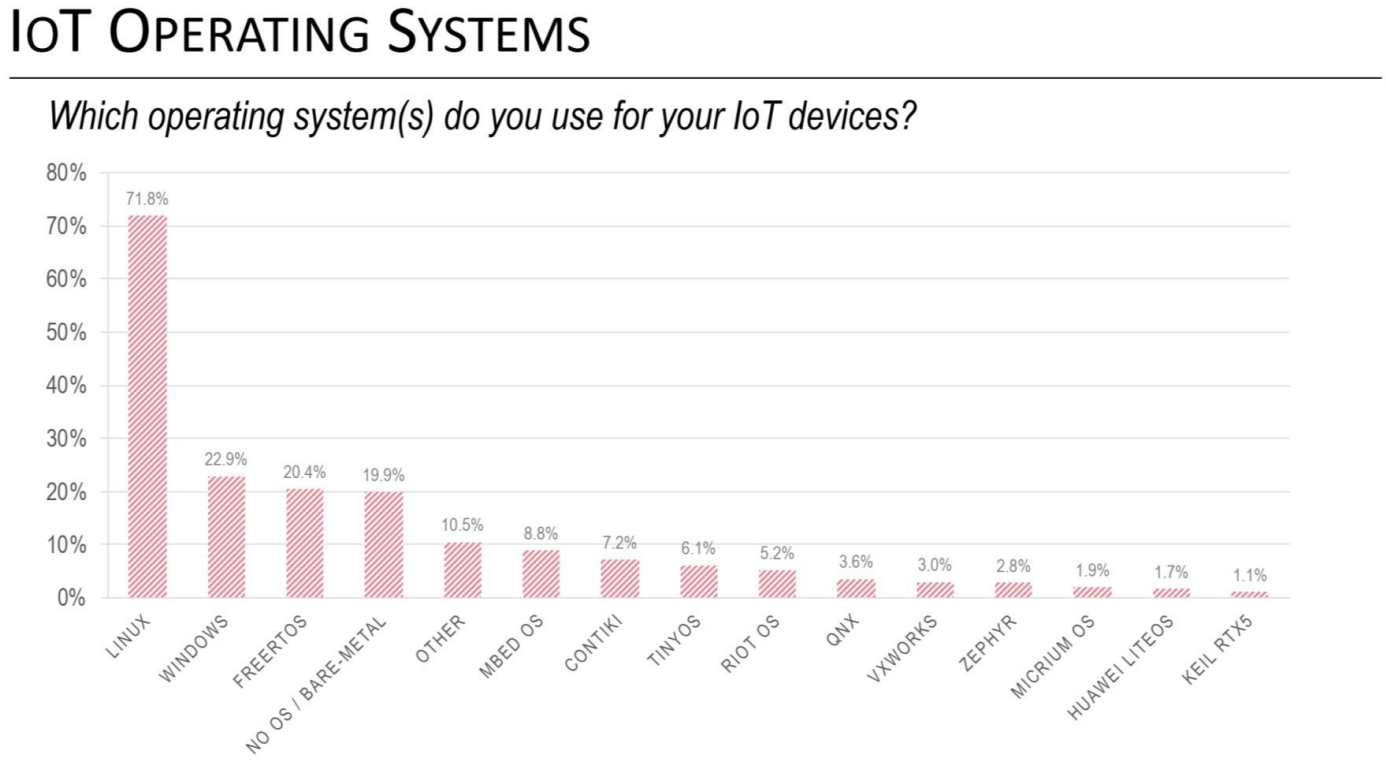
ATmega 328p
- 16 MHz
- 32 kB flash
- 2 kB RAM

STM32L476RG
- 80 MHz
- 1 MB flash
- 128 kB RAM

ESP-WROOM-32
- 240 MHz (dual core)
- 4 MB flash
- 520 kB RAM

Super Loop → RTOS

RTOS mas utilizados (2018)



Tipos de RTOS

Hard RTOS

En RTOS duros, el plazo se maneja muy estrictamente lo que significa que la tarea dada debe comenzar a ejecutar el plazo programado especificado, y debe completarse dentro de la duración asignada.
Ejemplo: Sistema de cuidados críticos médicos, sistemas de aeronaves, etc.

Firm RTOS

Este tipo de RTOS también tiene que seguir los plazos. Sin embargo, perder un plazo puede no tener un gran impacto, pero podría causar afectaciones no deseadas, como una enorme reducción en la calidad de un producto.
Ejemplo: Diversos tipos de aplicaciones Multimedia.

Soft RTOS

Tiempo suave Real RTOS, acepta algunos retrasos por parte del sistema operativo. En este tipo de RTOS, hay un plazo asignado para un trabajo específico, pero un retraso por una pequeña cantidad de tiempo es aceptable. Por lo tanto, los plazos son manejados suavemente por este tipo de RTOS.

Porque utilizar RTOS??

Diseñar un código teniendo en cuenta los tiempos y ejecuciones determinados implica un mayor trabajo para el programador.

Utilizar multitareas, es decir un proceso puede ejecutarse en simultaneo con otro sin suspender su ejecución

Por eso al utilizar un OS el sistema es más confiable, un hilo no depende de la ejecución de otro.

El uso de RTOS contribuye al determinismo del sistema, es decir los tiempos son establecidos y conocidos.

Podemos asignar un esquema de prioridades, de modo que las altas prioridades se ejecutarancuando esten listas

Desventajas

El SO requerira de por si solo más memoria de programa (flash)
Un RTOS requerirá extra RAM
Tendremos más consumo de CPU

freeRTOS

FreeRTOS es propiedas de Amazon (AWS). Es abierto con Licencia MIT. es una licencia de software libre y de código abierto que deja usar, cambiar y compartir el software con total libertad, incluso para proyectos comerciales. Es conocida por ser muy flexible, con muy pocas limitaciones, y valorada por su sencillez.

ESP32. Led Blink loop

```
void setup() {

  pinMode(18, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(18, HIGH); // turn the LED on
  delay(500);             // wait for 500 milliseconds
  digitalWrite(18, LOW);  // turn the LED off
  delay(500);             // wait for 500 milliseconds
}
```

FreeRTOS ESP32. Led Blink con RTOS

```
void toggleLED_1(void *parameter) {
  while(1) {
    digitalWrite(18, HIGH);
    vTaskDelay(500);
    digitalWrite(18, LOW);
    vTaskDelay(500);
  }
}

void loop() {
  // Do nothing
  // setup() and loop() run in their own task with priority 1 in core 1
  // on ESP32
}
```

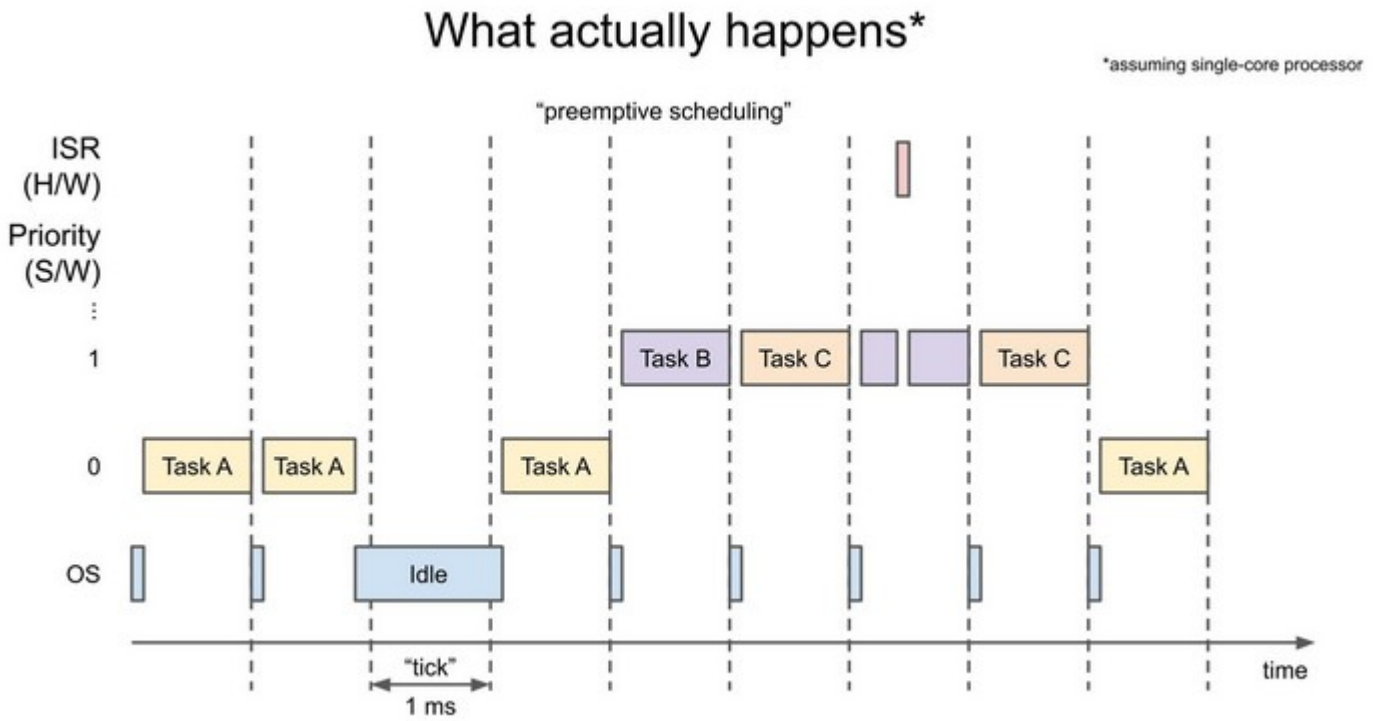
FreeRTOS ESP32. Led Blink con RTOS

```
void setup() {

  // Configure pin
  pinMode(led_pin, OUTPUT);

  // Task to run forever
  xTaskCreatePinnedToCore(  // Use xTaskCreate() in vanilla FreeRTOS
              toggleLED_1,  // Function to be called
              "Toggle 1",   // Name of task
              1024,         // Stack size (bytes in ESP32, words in FreeRTOS)
              NULL,         // Parameter to pass to function
              1,            // Task priority (0 to configMAX_PRIORITIES - 1)
              NULL,         // Task handle
              app_cpu);     // Run on one core for demo purposes (ESP32 only)
}
```

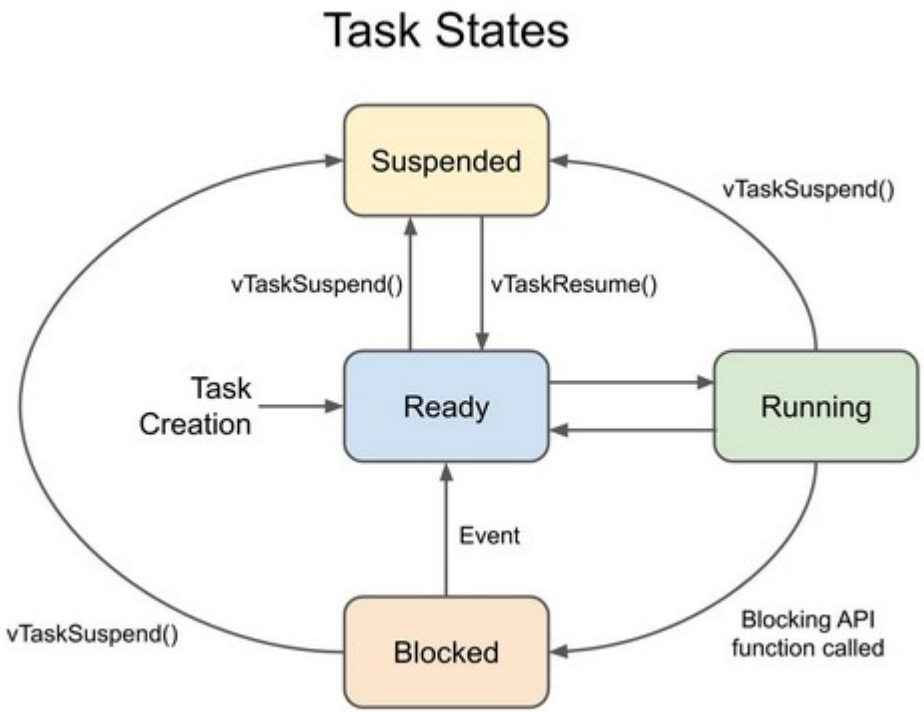
FreeRTOS ESP32. Led Blink con RTOS

```
void setup() {

  // Configure pin
  pinMode(led_pin, OUTPUT);

  // Task to run forever
  xTaskCreatePinnedToCore(  // Use xTaskCreate() in vanilla FreeRTOS
              toggleLED_1,  // Function to be called
              "Toggle 1",   // Name of task
              1024,         // Stack size (bytes in ESP32, words in FreeRTOS)
              NULL,         // Parameter to pass to function
              1,            // Task priority (0 to configMAX_PRIORITIES - 1)
              NULL,         // Task handle
              app_cpu);     // Run on one core for demo purposes (ESP32 only)
}
```

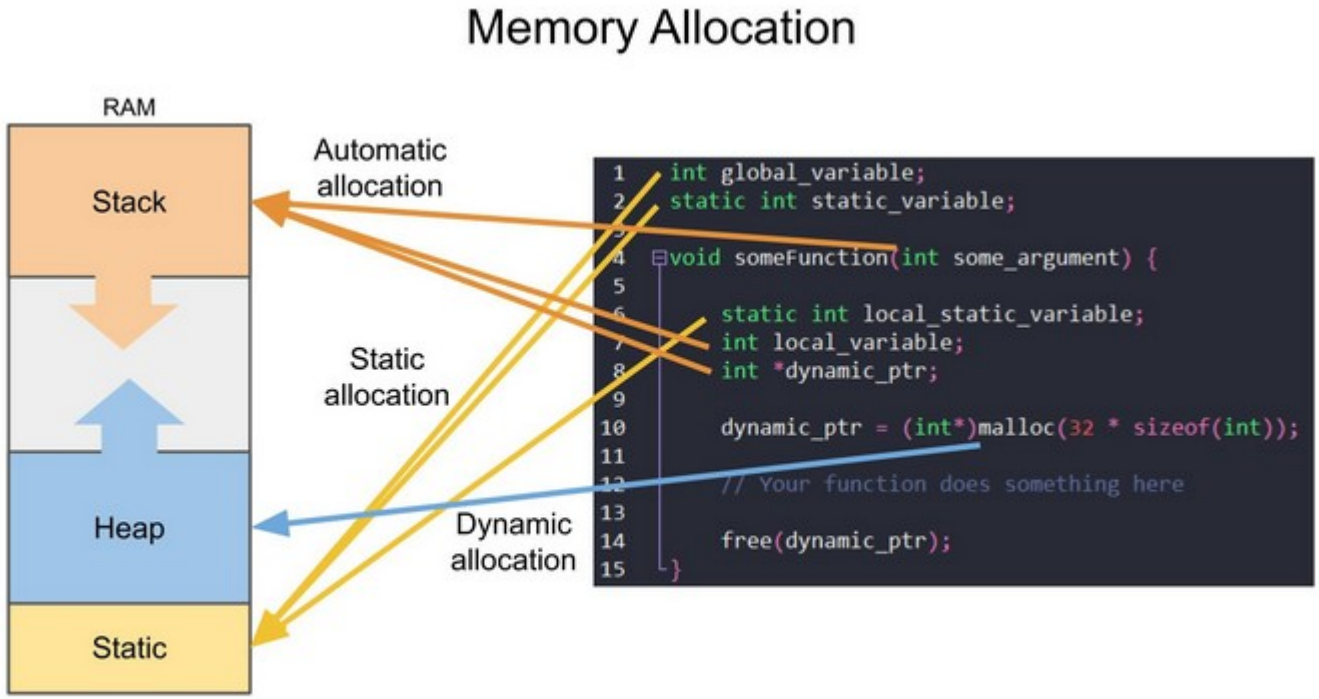
FreeRTOS. Temporizacion de tareas (Task Scheduling)



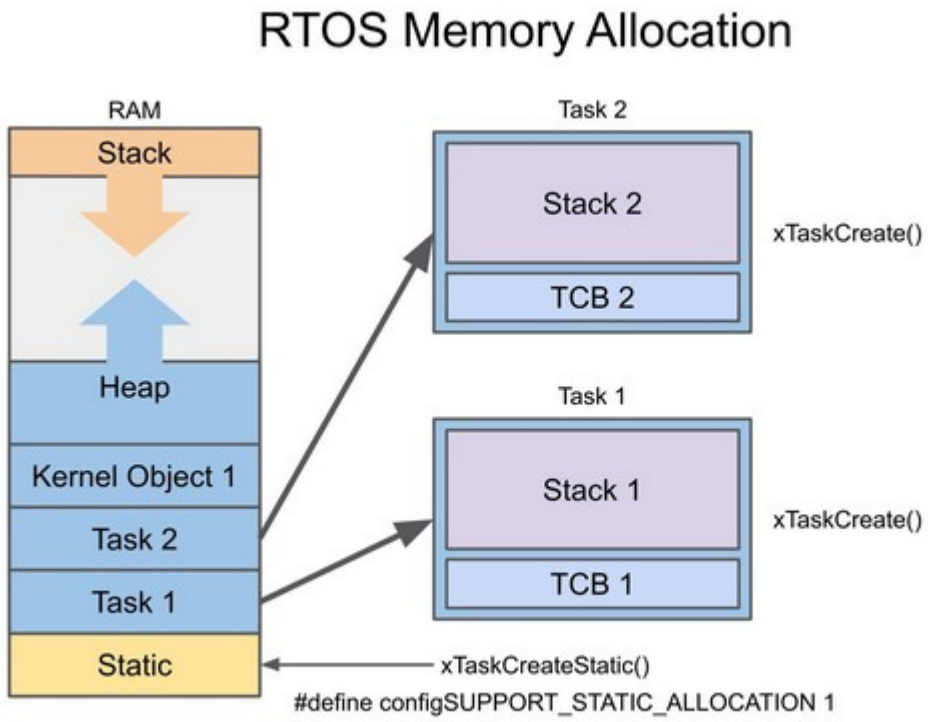
FreeRTOS. Temporizacion de tareas (Task Scheduling)



RTOS: Uso de memoria. Microcontrolador sin RTOS



RTOS: Uso de memoria. Microcontrolador con RTOS



¿Qué es una variable atómica en programación?

Una variable atómica en programación es una variable que permite realizar operaciones de forma atómica, es decir indivisible e ininterrumpida. Garantiza la coherencia e integridad de los datos compartidos entre los subprocesos de un programa multiproceso. Las variables atómicas se utilizan normalmente para implementar operaciones de lectura/modificación/escritura sin requerir bloqueo o sincronización explícitos.

Compartiendo datos

- Comunicación entre tareas:

  enfoque más sencillo: estructura/variable compartida
- Exclusión mutua

  asegurar acceso exclusivo para evitar datos corruptos
- Métodos:

  Deshabilitar interrupciones

  Operación Test&Set atómica (eg. 68000)

  Deshabilitar scheduling

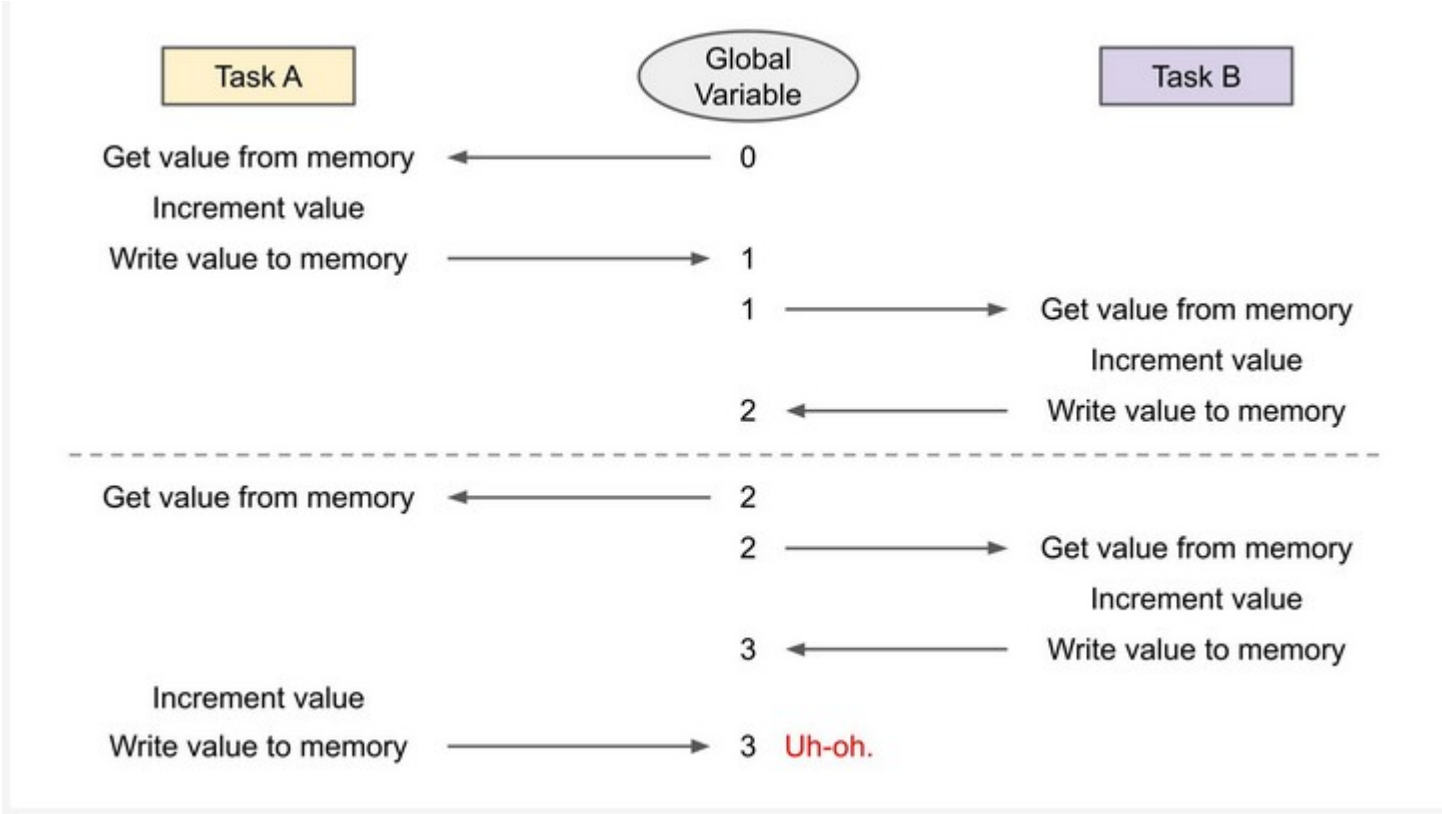
  Semáforos...

Semaforos

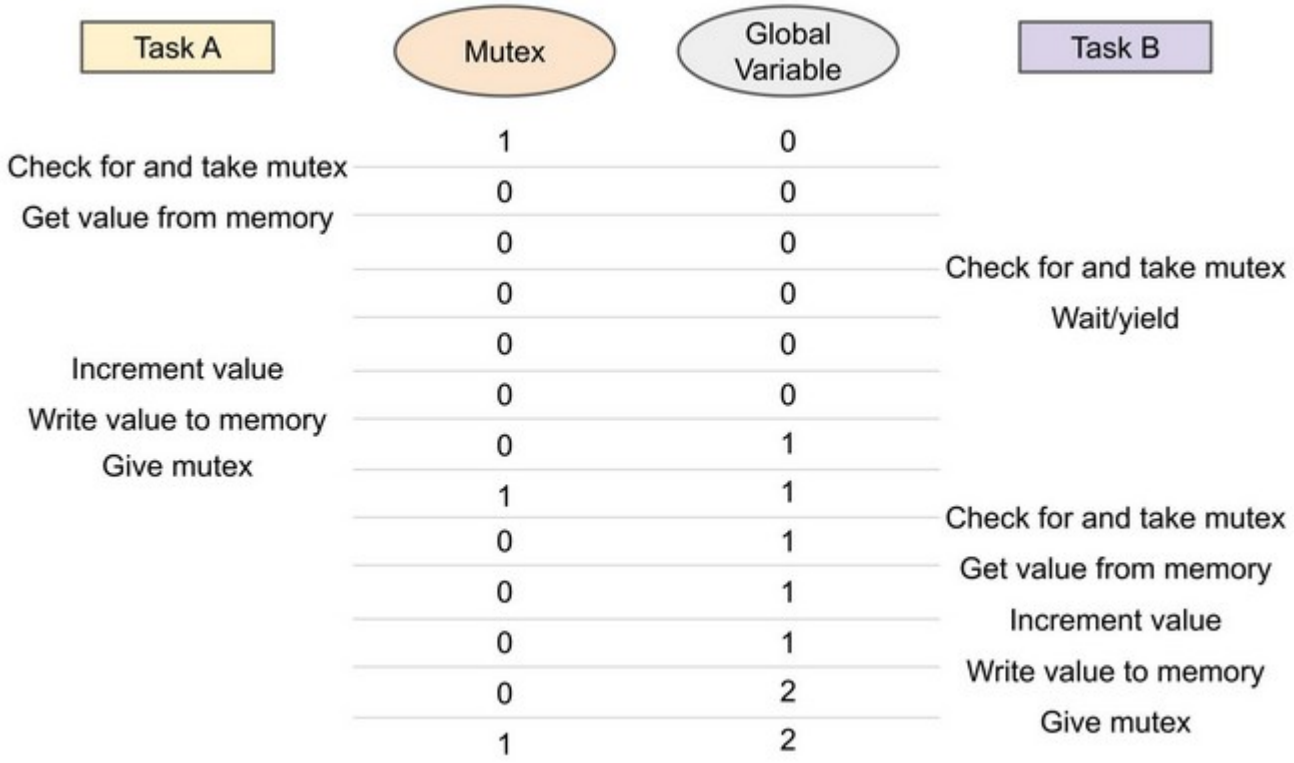
- Nomenclatura:
  - Take–Release (posta, llave)
  - Raise–Lower (semáforo ferroviario)
  - Wait–Signal
  - Pend–Post (esperar-avisar)

- Semáforos para exclusión mutua (MutEx)
  - proteger recursos compartidos: datos compartidos
- Semáforos para comunicación entre tareas.

Semaforo: MutEX. binario



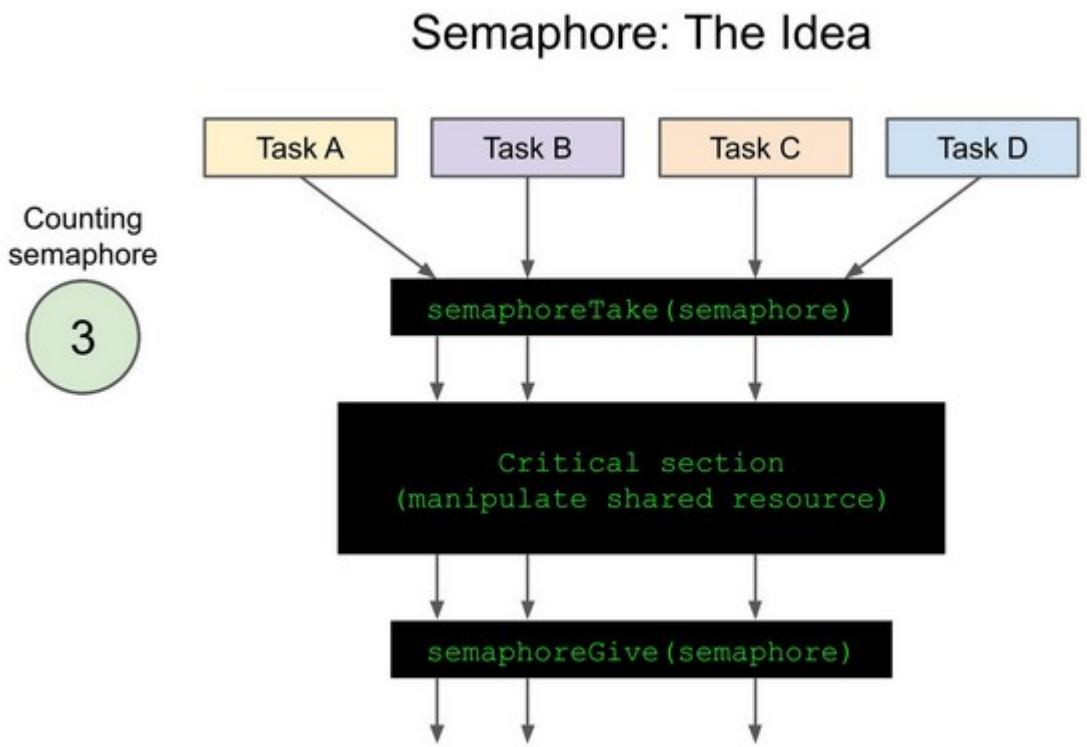
Semaforo: MutEX. Take / Give



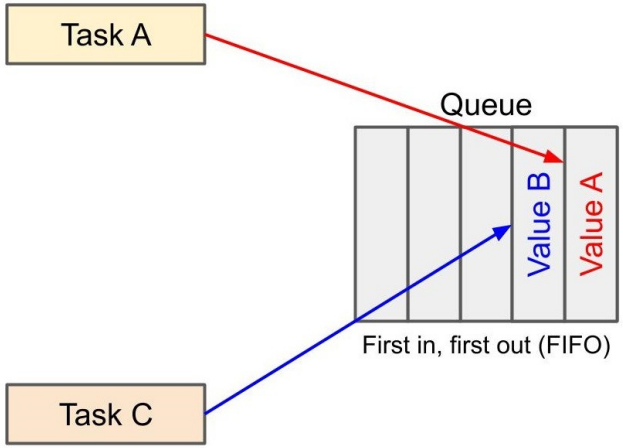
Semaforo: MutEX

```
// Globals
static SemaphoreHandle_t mutex;


// Create mutex before starting tasks
mutex = xSemaphoreCreateMutex();


// Take the mutex
xSemaphoreTake(mutex, portMAX_DELAY);


// Release the mutex so that the
creating function can finish
xSemaphoreGive(mutex)
```

Semaforo


Semaphore: The Idea
Task A
Task B
Task C
Task D
Counting semaphore
3
semaphoreTake(semaphore)
Critical section
(manipulate shared resource)
semaphoreGive(semaphore)

FreeRTOS.Listas de espera (Queue)



Se utilizan para transferir información entre tareas sin sobrescribir datos de otras tareas ni entrar en una condición de carrera. Una cola es un sistema FIFO (primero en entrar, primero en salir) donde los elementos se eliminan de la cola una vez leídos.

Ejemplos:

```
// Globals
static QueueHandle_t delay_queue;
static QueueHandle_t msg_queue;
```

```
if (xQueueReceive(msg_queue, (void *)&rcv_msg, 0) == pdTRUE) {
  Serial.print(rcv_msg.body);
  Serial.println(rcv_msg.count);
}
```

```
if (xQueueSend(delay_queue, (void *)&led_delay, 10) != pdTRUE) {
    Serial.println("ERROR: Could not put item on delay queue.");
}
```

Fin. gracias